



**Presentation topic:** Polymorphism

**Subject:** Object-Oriented programming

**Assigned by:** Engr: Jamsher

**Submitted by:** 20CS054

20CS058

20CS070

# Introduction

## **Polymorphism**

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.



In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.



## Types of polymorphism


In Java polymorphism is mainly divided into two types:

1. Compile-time Polymorphism
2. Runtime Polymorphism

### **Type 1:** Compile-time polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

**Note:** *But Java doesn't support the Operator Overloading.*



**Method Overloading:** When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments.

### Example 1 :

// Java program for Method Overloading  
// by Using Different Numbers of Arguments

// Class 1

// Helper class

class Helper {

    // Method 1

    // Multiplication of 2 numbers

    static int Multiply(int a, int b)

    {

        // Return product

        return a \* b;

    }



```
// Method 2
```

```
// // Multiplication of 3 numbers  
static int Multiply(int a, int b, int c)  
{
```

```
    // Return product  
    return a * b * c;
```

```
    }  
}
```

```
// Class 2
```

```
// Main class
```

```
class GFG {
```

```
    // Main driver method
```

```
    public static void main(String[] args)  
    {
```

```
        // Calling method by passing
```

```
        // input as in arguments
```

```
        System.out.println(Helper.Multiply(2, 4));
```

```
        System.out.println(Helper.Multiply(2, 7, 3));
```

```
    }  
}
```

**Output:** 8 & 42



## Type 2: Runtime polymorphism

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

### Example:

// Java Program for Method Overriding

// Class 1

// Helper class

class Parent {

    // Method of parent class

    void Print()

    {

        // Print statement

        System.out.println("parent class");

    }



```
}
```

```
// Class 2
```

```
// Helper class
```

```
class subclass1 extends Parent {
```

```
    // Method
```

```
    void Print() { System.out.println("subclass1"); }
```

```
}
```

```
// Class 3
```

```
// Helper class
```

```
class subclass2 extends Parent {
```

```
    // Method
```

```
    void Print()
```

```
{
```


```
        // Print statement
```

```
        System.out.println("subclass2");
```

```
}
```

```
}
```





```
// Class 4
// Main class
class GFG {
```

```
    // Main driver method
    public static void main(String[] args)
    {

        // Creating object of class 1
        Parent a;

        // Now we will be calling print methods
        // inside main() method

        a = new subclass1();
        a.Print();

        a = new subclass2();
        a.Print();

    }
}
```

```
Output:
Subclass1
subclass2
```



## Output explanation:

Here in this program, When an object of child class is created, then the method inside the child class is called. This is because The method in the parent class is overridden by the child class. Since The method is overridden, This method has more priority than the parent method inside the child class. So, the body inside the child class is executed.

